Best Practices for Networking Programming

Session 133

These are confidential sessions—please refrain from filming or taking pictures.

Application Technologies

Best Practices for Networking Programming

Getting from prototype to deployment

Becky Willrich
Technical Lead, URL Technologies

http://developer.apple.com/wwdc2005/

Introduction

- You finished the prototype…
- …and the boss loves it…
- …so he wants it ready NEXT WEEK!

- How do you get the networking code ready for deployment?

Getting to Deployment

- Networking code needs to be ready for a variety of network configurations
  - Slow connections
  - Misconfigured networks
  - Firewalls
  - Authenticated servers

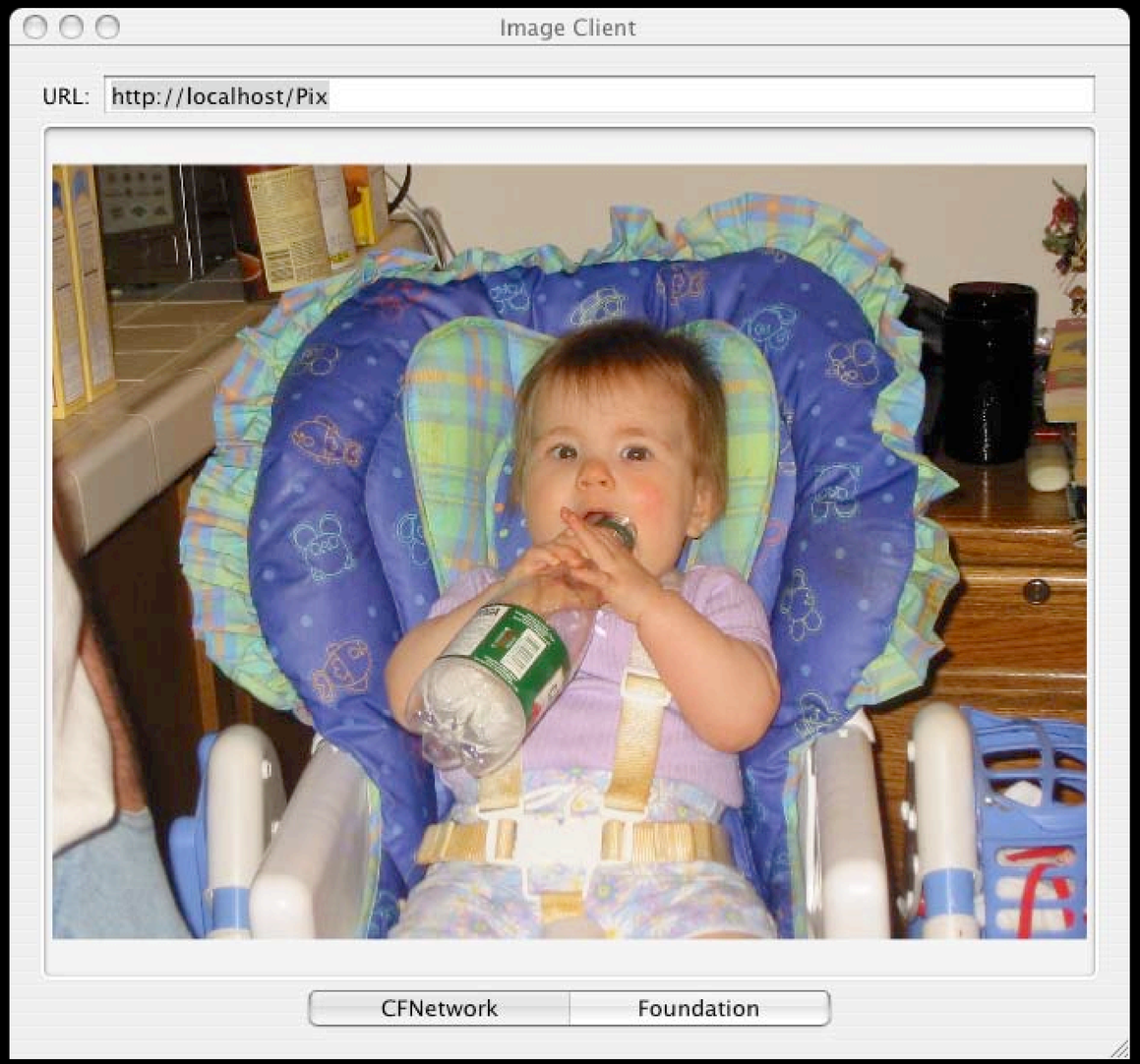
- We'll show you how to get there!

Getting to Deployment

- Slow connections
  - Use asynchronous downloads
- Misconfigured networks
  - Add net diagnostic support
- Firewalls
  - Support proxies
- Authenticated servers
  - Add authentication

Getting to Deployment

The Image Client application



Taking Image Client to Deployment

- Step 0—First prototype
- Step 1—Conversion to asynchronous networking
- Step 2—Diagnose network failures
- Step 3—Support firewalls
- Step 4—Support password-protected servers
- Step 5—Reusing passwords
- Step 6—Storing and retrieving passwords
- Step 7—Support password-protected firewalls

Supporting Resources

- In the sample code
  - One ImageClient project for each step of the way
  - Configuration files and instructions for setting up the servers you need
- Networking lab tomorrow morning

Related session

Lab: Networking APIs

Friday 9:00AM

The Image Client Application

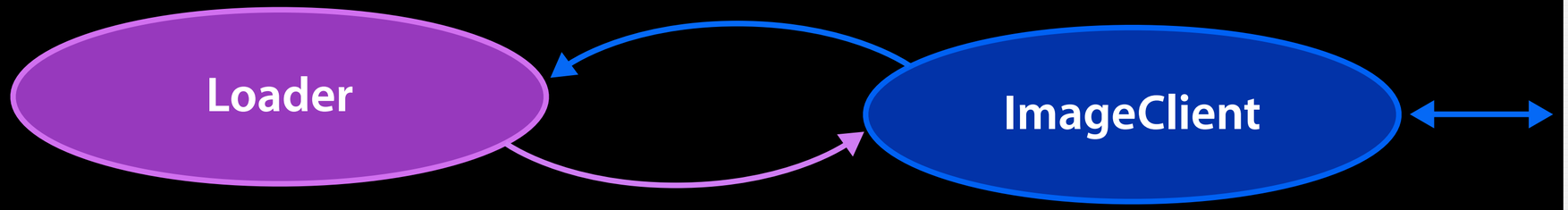
First prototype

http://developer.apple.com/wwdc2005/

The Image Client Application

Architecture

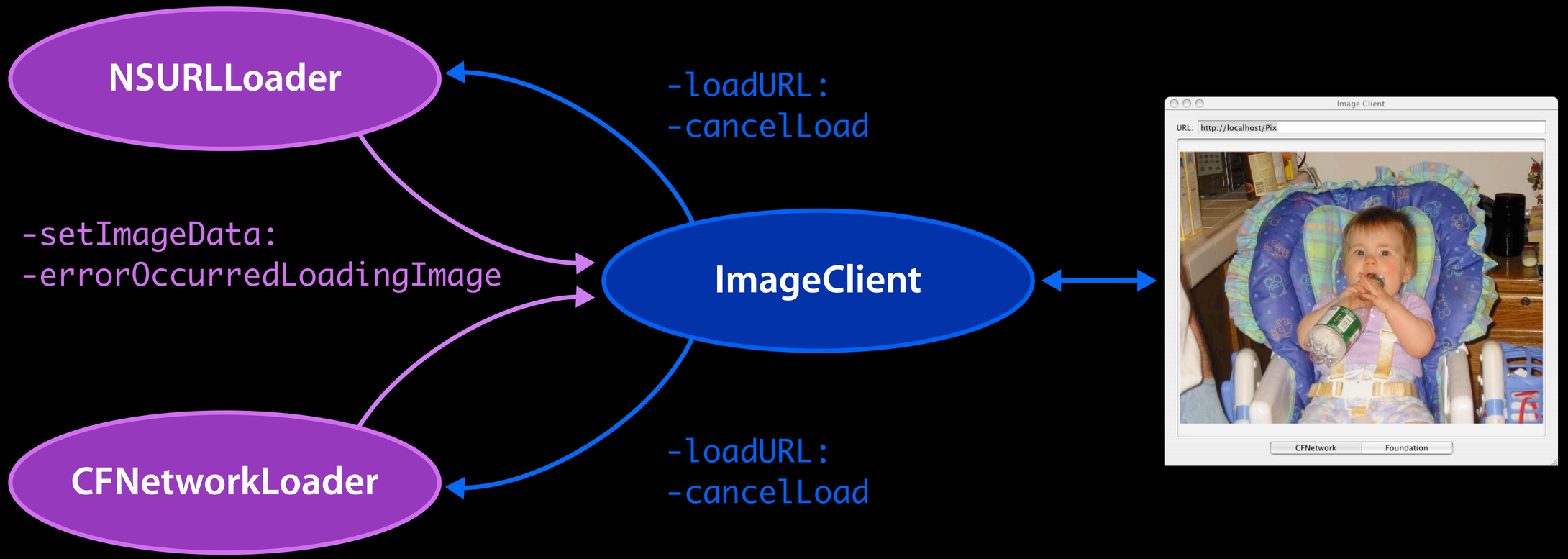
-loadURL:
-cancelLoad



-setImageData:
-errorOccurredLoadingImage

The Image Client Application

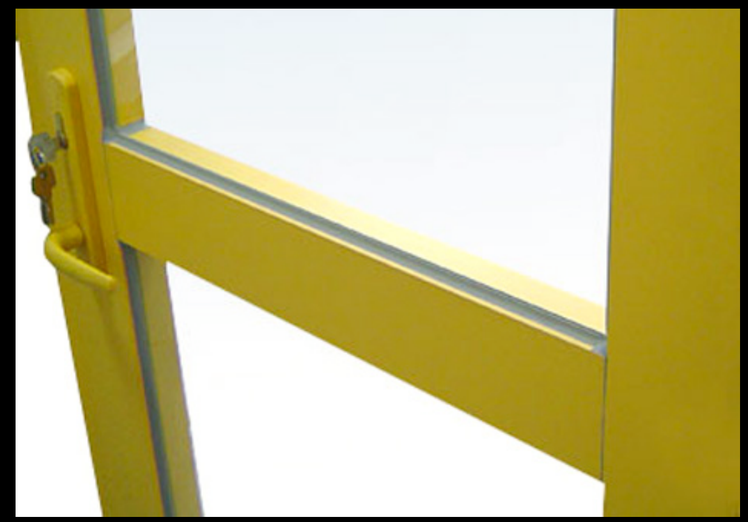
Architecture



CFNetwork vs. NSURL

Which library do I use?

- NSURL is more feature-rich
  - Automatic proxy support
  - Caching infrastructure
  - Authentication and credential infrastructure
  - Cookie integration



- CFNetwork is more flexible
  - Better threading control
  - Configurable proxy settings
  - More access to the details
  - No Objective-C dependency

Loading the URL

NSURL

```
- (void)loadURL:(NSURL *)url {
    NSData *imageData = [url resourceDataUsingCache:NO];
    if (imageData && [imageData length]) {
        [imageClient setImageData:imageData];
    } else {
        [imageClient errorOccurredLoadingImage];
    }
}
```

Loading the URL

CFNetwork

```
- (void)loadURL:(NSURL *)url {
    // set the request for the new transaction
    if (request) CFRelease(request);

    request = CFHTTPMessageCreateRequest(NULL, CFSTR("GET"),
        (CFURLRef)url, kCFHTTPVersion1_1);

    // Start the load
    [self loadRequest];
}
```

Loading the URL

CFNetwork

```
- (void)loadRequest {
    ...
    readStream = CFReadStreamCreateForHTTPRequest(NULL, request);

    if (CFReadStreamOpen(readStream)) {
        do {
            int bytesRead = CFReadStreamRead(readStream, buf, BUFSIZE);
            if (bytesRead > 0) {
                CFDataAppendBytes(data, buf, bytesRead);
            } else if (bytesRead == 0) {
                // All done
                [imageClient setImageData:(NSData *)data];
                done = YES;
            } else {
                // Error occurred
                [imageClient errorOccurredLoadingImage];
                done = YES;
            }
        } while (!done);
    }
    ...
```

Loading the URL

CFNetwork

```
- (void)loadRequest {
    ...

    // Clean up
    CFReadStreamClose(readStream);
    CFRelease(readStream);
    readStream = NULL;

    ...
}
```

Problems

- Seems to hang when used over a modem line
- No way to tell what’s gone wrong
- Can’t use it behind a firewall
- Can’t use it on a password-protected server

Step 1: Adding Asynchrony

Integrating with the Run Loop

http://developer.apple.com/wwdc2005/

Why Asynchronous?

- Avoid the spinning wheel of doom
- Allow user actions to take place during potentially lengthy download
- And remember…

…ANY Download Could Be Lengthy!

- Maybe you're connected by modem
- Maybe the user's downloading 10 GB of pictures
- Maybe the DNS server is down
- Maybe the server is overloaded

Why the Run Loop?

- Avoids an extra thread in almost all cases*
- Avoids wasted work from “busy waiting” like polling
- Integrates well with other event-based activity

* Extremely high bandwidth situations may need a second thread

Adding Asynchrony—NSURLLoader

Creating an NSURLConnection

```
- (void)loadURL:(NSURL *)url {
    // Cancel any load currently in progress
    [self cancelLoad];

    // Create an empty data to receive bytes from the download
    data = [[NSMutableData alloc] init];

    // Start the download
    NSURLRequest *request = [NSURLRequest requestWithURL:url];
    connection = [[NSURLConnection alloc] initWithRequest:request
delegate:self];
}
```

Adding Asynchrony—NSURLLoader

Handling the delegate messages

```
- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData
*)newData {
    [data appendData:newData];
}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection {
    [imageClient setImageData:data];
    [self cancelLoad]; // This shuts down the connection for us
}

- (void)connection:(NSURLConnection *)connection didFailWithError:
(NSError *)error {
    [imageClient errorOccurredLoadingImage];
    [self cancelLoad]; // This shuts down the connection for us
}
```

Adding Asynchrony—NSURLLoader

Canceling an NSURLConnection

```
- (void)cancelLoad {
    if (connection) {
        [connection cancel];
        [connection release];
        connection = nil;
    }
    if (data) {
        [data release];
        data = nil;
    }
}
```

Adding Asynchrony—CFNetworkLoader

Setting up the stream for asynchronous use

```
- (void)loadRequest {
    ...
    readStream = CFReadStreamCreateForHTTPRequest(NULL, request);

    CFStreamClientContext context = {0, self, NULL, NULL, NULL};
    CFReadStreamSetClient(readStream,
                          kCFStreamEventHasBytesAvailable |
                             kCFStreamEventEndEncountered |
                              kCFStreamEventErrorOccurred,
                          handleStreamEvent,
                          &context);
    CFReadStreamScheduleWithRunLoop(readStream,
                                    CFRunLoopGetCurrent(),
                                    kCFRunLoopCommonModes);
    CFReadStreamOpen(readStream);
}
```

Adding Asynchrony—CFNetworkLoader

Handling the callback

```
- (void)loadRequest {
    ...
    readStream = CFReadStreamCreateForHTTPRequest(NULL, request);

    if (CFReadStreamOpen(readStream)) {
        do {
            int bytesRead = CFReadStreamRead(readStream, buf, BUFSIZE);
            if (bytesRead > 0) {
                CFDataAppendBytes(data, buf, bytesRead);
            } else if (bytesRead == 0) {
                // All done
                [imageClient setImageData:(NSData *)data];
                done = YES;
            } else {
                // Error occurred
                [imageClient errorOccurredLoadingImage];
                done = YES;
            }
        } while (!done);
    }
    ...
```

HasBytesAvailable

EndEncountered

ErrorOccurred

Adding Asynchrony—CFNetworkLoader

Handling the callback

```
- (void)handleStreamEvent:(CFStreamEventType)event {
    switch (event) {
        case kCFStreamEventHasBytesAvailable: {
            UInt8 buffer[BUFSIZE];
            int bytesRead = CFReadStreamRead(readStream, buf, BUFSIZE);
            if (bytesRead > 0) {
                CFDataAppendBytes(data, buf, bytesRead);
            }
            break;
        }
        case kCFStreamEventEndEncountered:
            [imageClient setImageData:(NSData *)data];
            [self cancelLoad];
            break;
        case kCFStreamEventErrorOccurred:
            [imageClient errorOccurredLoadingImage];
            [self cancelLoad];
            break;
        default: ...
    }
}
```

Adding Asynchrony—CFNetworkLoader

Cleaning up

```
- (void)loadRequest {
    ...

    // Clean up
    CFReadStreamClose(readStream);
    CFRelease(readStream);
    readStream = nil;

    ...
}
```

cancelLoad

Adding Asynchrony—CFNetworkLoader

Cleaning up

```
- (void)cancelLoad {

  if (readStream) {
    CFReadStreamClose(readStream);
    CFReadStreamSetClient(readStream, kCFStreamEventNone, NULL, NULL);
    CFReadStreamUnscheduleFromRunLoop(readStream,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);

    CFRelease(readStream);
    readStream = NULL;
  }
  ...
}
```

Demo

Asynchronous ImageClient

http://developer.apple.com/wwdc2005/

Step 2: Adding Network Diagnostics

Using CFNetDiagnostics to diagnose error conditions

http://developer.apple.com/wwdc2005/

What is CFNetDiagnostics?

- Networks are
  - Easy to misconfigure
  - Fragile
  - Hard to debug
- CFNetDiagnostics is an API to call when something network-y goes wrong

Using CFNetDiagnostics

- Create after some kind of networking failure
  - CFNetDiagnosticCreateWithStreams() from a failed CFStream
  - CFNetDiagnosticCreateWithURL() from an unreachable URL
- Display a diagnostic string to the user
  - CFNetDiagnosticCopyNetworkStatusPassively()
- Guide the user to a UI to diagnose the problem
  - CFNetDiagnosticDiagnoseProblemInteractively()

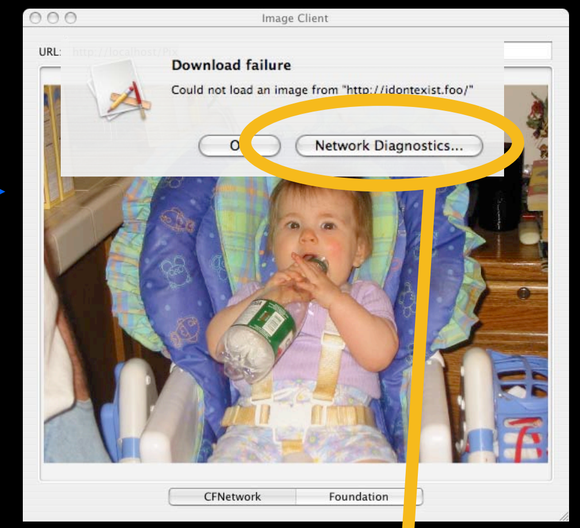
Handling Authentication

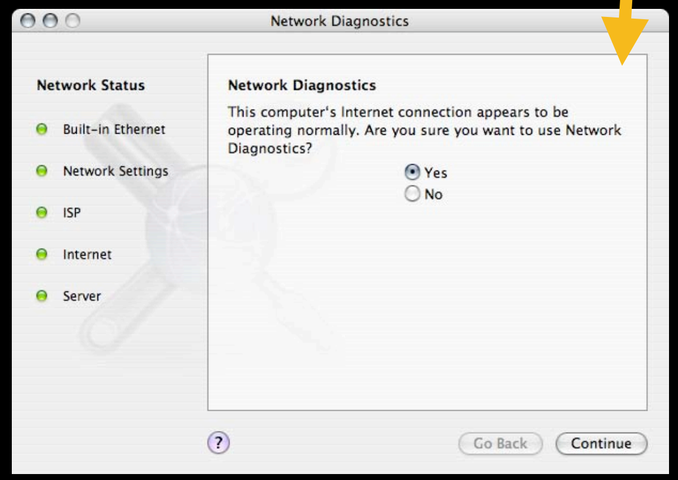
Architecture

-errorOccuredLoadingImage:
(CFNetDiagnosticsRef)diag

Loader

ImageClient





Network Diagnostics—NSURLLoader

Creating the CFNetDiagnostic object

```
- (void)loadURL:(NSURL *)theURL {
    ...
    if (url) [url release];
    url = theURL;
    [url retain];
    ...
}

- (void)connection:(NSURLConnection *)connection didFailWithError:
(NSError *)error {
    CFNetDiagnosticRef diagnostics =
        CFNetDiagnosticCreateWithURL(NULL, (CFURLRef)url);
    [imageClient errorOccurredLoadingImage:diagnostics];
    [self cancelLoad]; // This shuts down the connection for us
}
```

Network Diagnostics—CFNetworkLoader

Creating the CFNetDiagnostic object

```
case kCFStreamEventErrorOccurred: {
  CFNetDiagnosticRef diagnostics =
    CFNetDiagnosticCreateWithStreams(NULL, readStream, NULL);
  [imageClient errorOccurredLoadingImage:diagnostics];
  [self cancelLoad];
  break;
}
```

Network Diagnostics—ImageClient

Bringing up the interactive UI

```
- (void)errorSheetDidEnd:(NSWindow *)sheet returnCode:(int)returnCode
contextInfo:(void *)contextInfo {
  CFNetDiagnosticRef diagnostics = (CFNetDiagnosticRef)contextInfo;

  if (returnCode == NSAlertDefaultReturn) {
    (void)CFNetDiagnosticDiagnoseProblemInteractively(diagnostics);
  }
  CFRelease(diagnostics);
}
```

Demo

Network Diagnostics

http://developer.apple.com/wwdc2005/

Step 3: Navigating Firewalls

Adding proxy support

http://developer.apple.com/wwdc2005/

Navigating Firewalls

- Configure firewalls by setting proxies
- Users go to Network Preferences and put proxies there
- The proxy information is stored in System Configuration
- API access through SystemConfiguration.framework
  - SCDynamicStoreCopyProxies() retrieves the current proxy settings
  - SCDynamicStores notify you when the proxy changes

Navigating Firewalls—NSURLLoader

There is no step three!

- NSURLConnection does the work for you
  - Always applies the current proxy settings
  - Monitors for changes to those settings

Managing Firewalls—CFNetworkLoader

The quick-and-dirty way

```
- (void)loadRequest {

    ...

    readStream = CFReadStreamCreateForHTTPRequest(NULL, request);

    CFDictionaryRef proxyDict = SCDynamicStoreCopyProxies(NULL);
    CFReadStreamSetProperty(readStream, kCFStreamPropertyHTTPProxy,
proxyDict);

    ...

    CFReadStreamOpen(readStream);
}
```

The Quick-and-Dirty Way

- It works, but is expensive
  - Recreate the SCDynamicStore every time
  - Rebuild the connection to System Configuration every time
  - No remembered state with System Configuration
- Appropriate for
  - One-off or rare downloads
  - Very lengthy downloads
- Inappropriate for
  - Performance-critical regions
  - Frequent downloads

The Robust Way

- Create a single, persistent SCDynamicStore
- Put it in the run loop to get notification of proxy changes
- Use it to get the latest proxy settings
- Update your copy of the proxies when told of changes
- Clean it up and dispose when through

Managing Firewalls—CFNetworkLoader

Creating the dynamic store

```
- (id)initWithImageClient:(ImageClient *)imgClient {
  ...

// Create the dynamic store, to monitor changes to the proxy dictionary
SCDynamicStoreContext context = {0, self, NULL, NULL, NULL};
systemDynamicStore = SCDynamicStoreCreate(NULL,
                        CFSTR("ImageClient"), proxyChanged, &context);

...
```

Managing Firewalls—CFNetworkLoader

Adding the dynamic store to the run loop

```
- (id)initWithImageClient:(ImageClient *)imgClient {
  ...

  // Set up the store to monitor any changes to the proxies
  CFStringRef proxiesKey = SCDynamicStoreKeyCreateProxies(NULL);
  CFArrayRef keyArray =
    CFArrayCreate(NULL, (const void **)(&proxiesKey), 1, &kCFTypeArrayCallBacks);
SCDynamicStoreSetNotificationKeys(systemDynamicStore, keyArray, NULL);
CFRelease(keyArray);
CFRelease(proxiesKey);

// Add the dynamic store to the run loop
CFRunLoopSourceRef storeRLSource =
  SCDynamicStoreCreateRunLoopSource(NULL, systemDynamicStore, 0);
CFRunLoopAddSource(CFRunLoopGetCurrent(), storeRLSource, kCFRunLoopCommonModes);
CFRelease(storeRLSource);

...
```

Managing Firewalls—CFNetworkLoader

Getting the latest proxy settings

```
- (id)initWithImageClient:(ImageClient *)imgClient {
    ...
    // Preload the proxy dictionary with the current settings
    proxyDictionary = SCDynamicStoreCopyProxies(systemDynamicStore);
  }
  return self;
}
```

Managing Firewalls—CFNetworkLoader

Updating when the proxy settings change

```
static void proxyChanged(SCDynamicStoreRef store, CFArrayRef changedKeys,
void *info) {
    CFNetworkLoader *self = (CFNetworkLoader *)info;
    [self proxyChanged];
}

// Callback for a change in the proxy dictionary
- (void)proxyChanged {
    CFRelease(proxyDictionary);
    proxyDictionary = SCDynamicStoreCopyProxies(systemDynamicStore);
}
```

Managing Firewalls—CFNetworkLoader

Using the proxy data

```
- (void)loadRequest {
  ...

  readStream = CFReadStreamCreateForHTTPRequest(NULL, request);

  CFReadStreamSetProperty(readStream,
    kCFStreamPropertyHTTPProxy, proxyDictionary);

  CFStreamClientContext context = {0, self, NULL, NULL, NULL};
  CFReadStreamSetClient(readStream, kCFStreamEventHasBytesAvailable |
kCFStreamEventEndEncountered | kCFStreamEventErrorOccurred,
handleStreamEvent, &context);
  CFReadStreamScheduleWithRunLoop(readStream, CFRunLoopGetCurrent(),
kCFRunLoopCommonModes);

    ...
```

Managing Firewalls—CFNetworkLoader

Cleaning up the dynamic store

```
- (void)dealloc {
    if (request) CFRelease(request);
    [self cancelLoad];

    if (proxyDictionary) CFRelease(proxyDictionary);

    // Need to invalidate the dynamic store's run loop source
    // to get the store out of the run loop
    CFRunLoopSourceRef rls = SCDynamicStoreCreateRunLoopSource(NULL,
systemDynamicStore, 0);
    CFRunLoopSourceInvalidate(rls);
    CFRelease(rls);
    CFRelease(systemDynamicStore);

    [super dealloc];
}
```

Demo

Proxy support in Image Client

http://developer.apple.com/wwdc2005/

Step 4: Handling Authentication

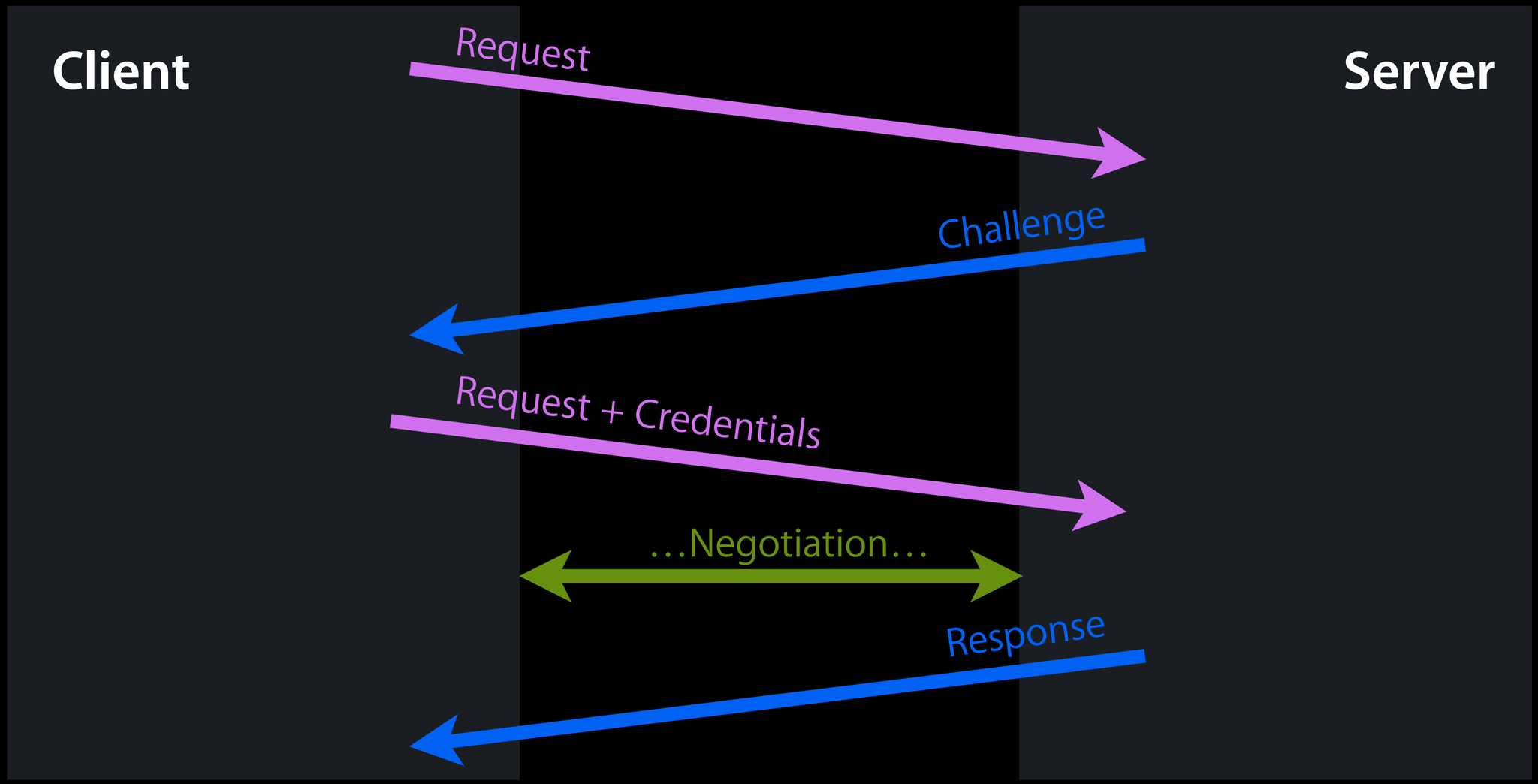
Recognizing challenges and using credentials

http://developer.apple.com/wwdc2005/

Handling Authentication

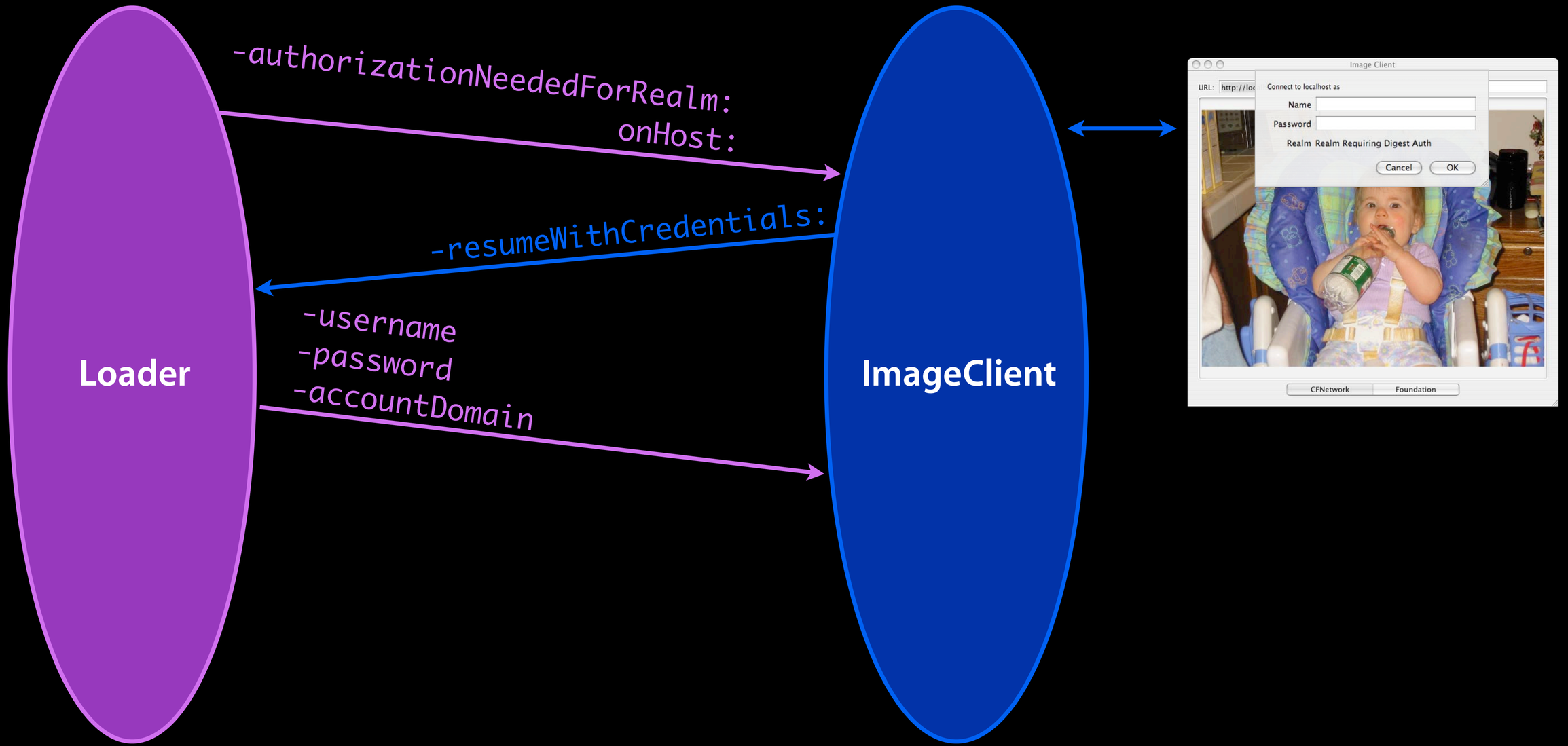
- Some servers require passwords
- Some proxies require passwords
- Your application needs to be ready

Handling Authentication



Handling Authentication

Architecture



Handling Authentication—NSURLLoader

Fielding the challenge

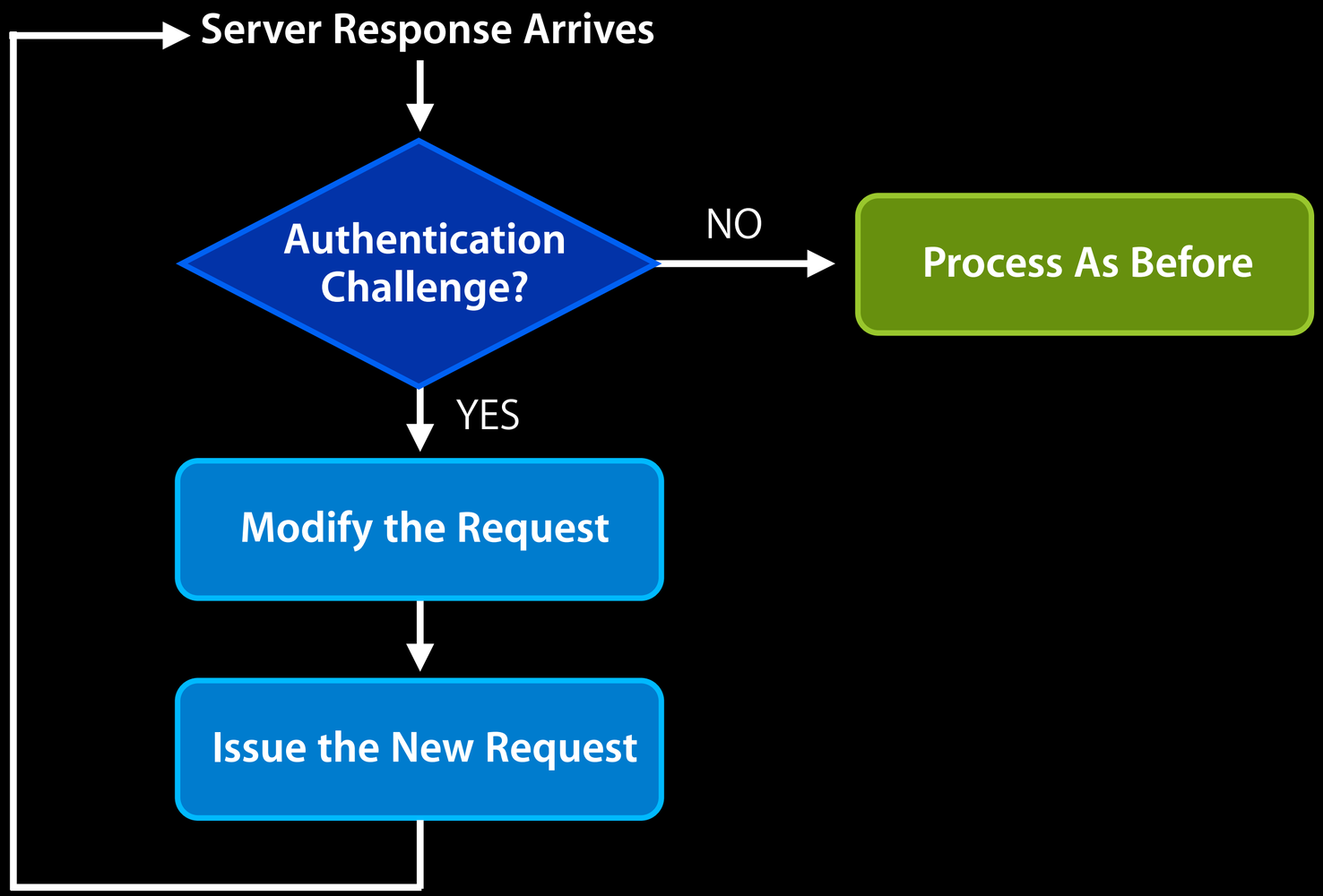
```
/* Called when the connection receives a challenge from the server */
- (void)connection:(NSURLConnection *)connection didReceiveAuthenticationChallenge:
(NSURLAuthenticationChallenge *)newChallenge {

    if (challenge) [challenge release];
    challenge = newChallenge;
    [challenge retain];

    NSURLProtectionSpace *space = [challenge protectionSpace];
    [imageClient authorizationNeededForRealm:[space realm] onHost:[space host]];
}

/* Called when a connection waiting for user/pass is cancelled, so that we can clean up any
lingering state */
- (void)connection:(NSURLConnection *)connection didCancelAuthenticationChallenge:
(NSURLAuthenticationChallenge *)challenge {
    [self cancelLoad];
}
```

Handling Authentication—NSURLLoader

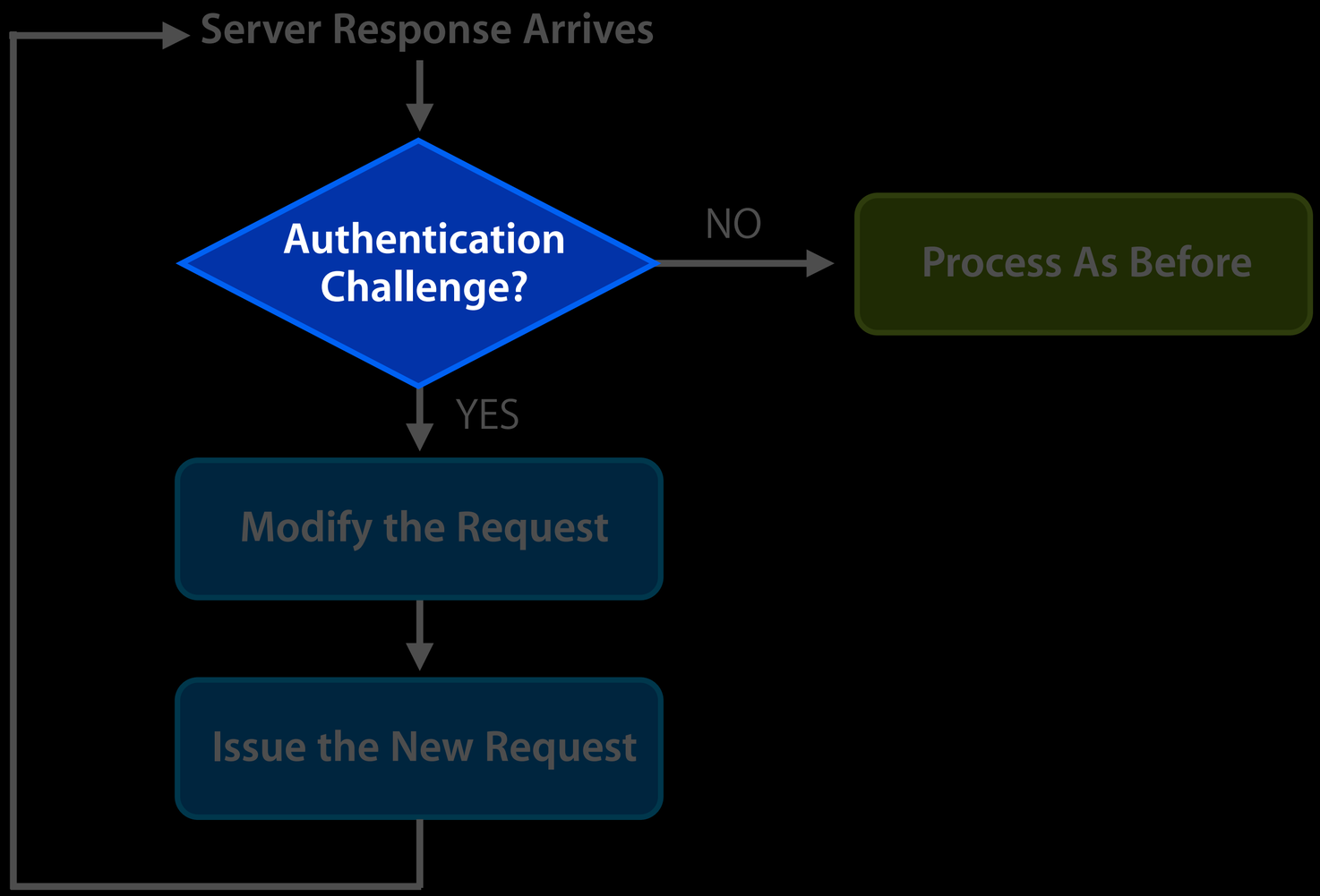
Resuming with credentials

```
- (void)resumeWithCredentials {
    // Grab the username and password from imageClient and apply them to the challenge
    NSString *user = [imageClient username];
    NSString *pass = [imageClient password];

    // Guarantee values
    if (!user) user = @"";
    if (!pass) pass = @"";

    NSURLCredential *creds = [NSURLCredential credentialWithUser:user password:pass
        persistence:NSURLCredentialPersistenceForSession];

    [[challenge sender] useCredential:creds forAuthenticationChallenge:challenge];

    [challenge release];
    challenge = nil;
}
```

Handling Authentication—CFNetworkLoader



Handling Authentication—CFNetworkLoader



Handling Authentication—CFNetworkLoader

Identifying authentication challenges

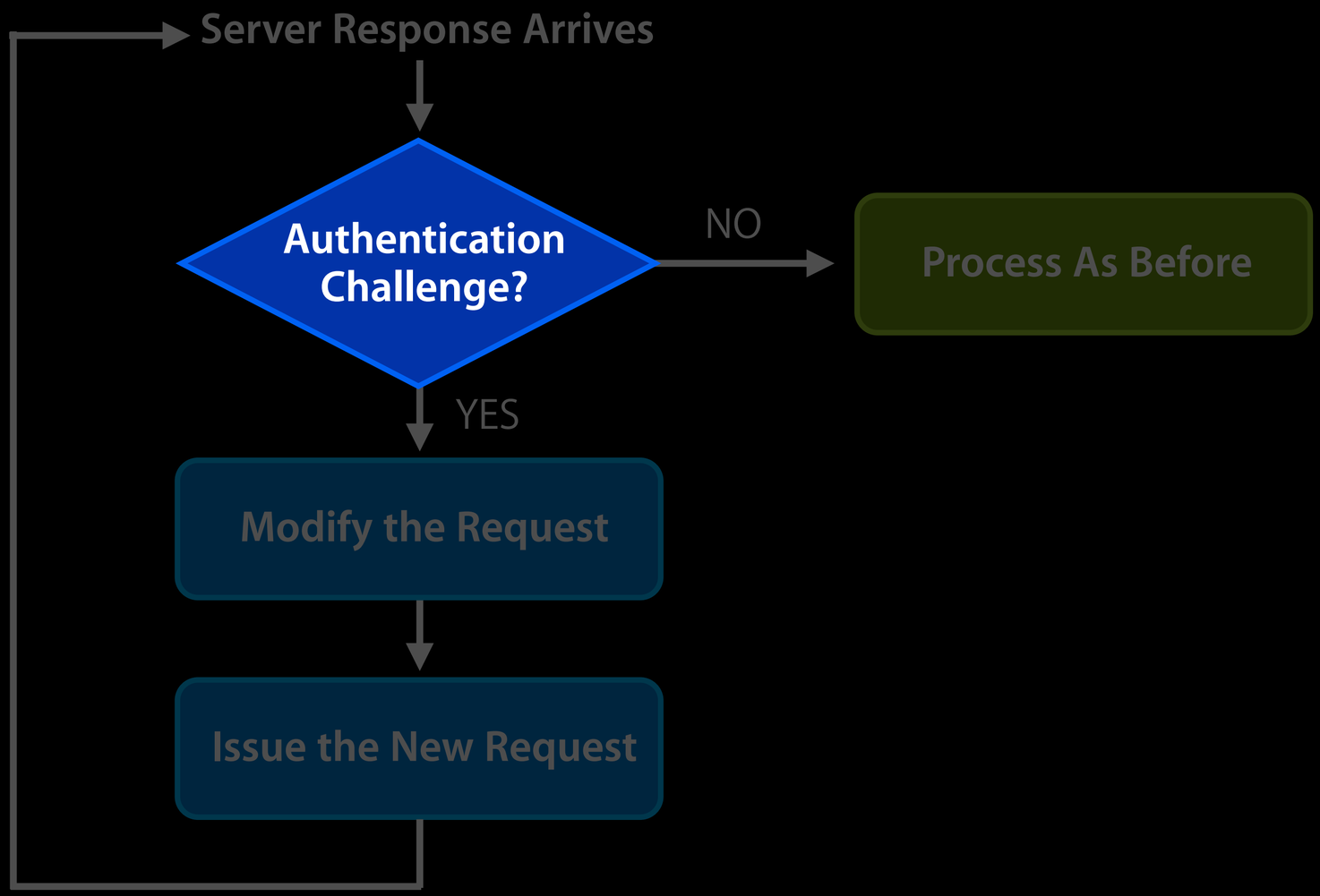
```
- (void)cancelLoad {
  ...
  haveExaminedHeaders = NO;
}

- (void)handleStreamEvent:(CFStreamEventType)event {
  ...
case kCFStreamEventHasBytesAvailable: {
  if (!haveExaminedHeaders) {
    haveExaminedHeaders = YES;
    if ([self isAuthorizationFailure]) {
      [self retryAfterAuthorizationFailure];
      break;
    }
  ...
case kCFStreamEventEndEncountered:
    if (!haveExaminedHeaders) {
    ...
```

Handling Authentication—CFNetworkLoader

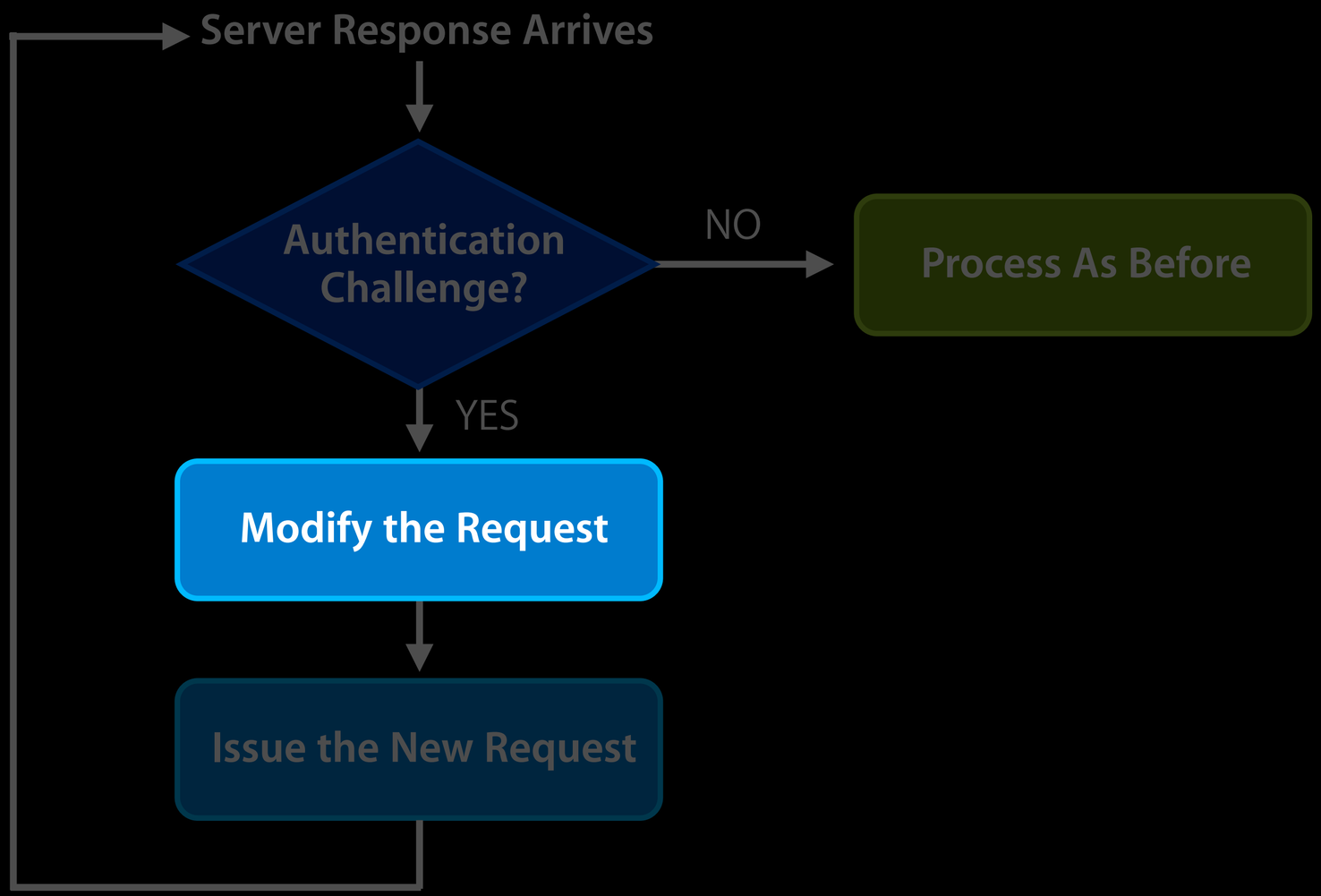
Identifying authentication challenges

```
- (BOOL)isAuthorizationFailure {
  CFHTTPMessageRef responseHeaders =
    (CFHTTPMessageRef)CFReadStreamCopyProperty(readStream,
       kCFStreamPropertyHTTPResponseHeader);

  if (responseHeaders) {
    // Is the server response an challenge for credentials?
    BOOL isAuthenticationChallenge =
       (CFHTTPMessageGetResponseStatusCode(responseHeaders) == 401);
    CFRelease(responseHeaders);
    return isAuthenticationChallenge;
  }
  return NO;
}
```

Handling Authentication—CFNetworkLoader



Handling Authentication—CFNetworkLoader



Modifying the Request

The quick-and-dirty way

```
CFHTTPMessageRef responseHeaders = <copy response header from stream>;
if (CFHTTPMessageAddAuthentication(request, responseHeaders,
        <username>, <password>, NULL, NO) {
    <retry the request>
} else {
    <failed to apply credentials>;
}
```

The Quick-and-Dirty Way

- No remembered state for multiple requests to a server
- No support for multi-leg authentication types
  - No support for NTLM!
- Poor performance

Not Suitable for Deployment Code

Modifying the Request—the Right Way

Use CFHTTPAuthentication

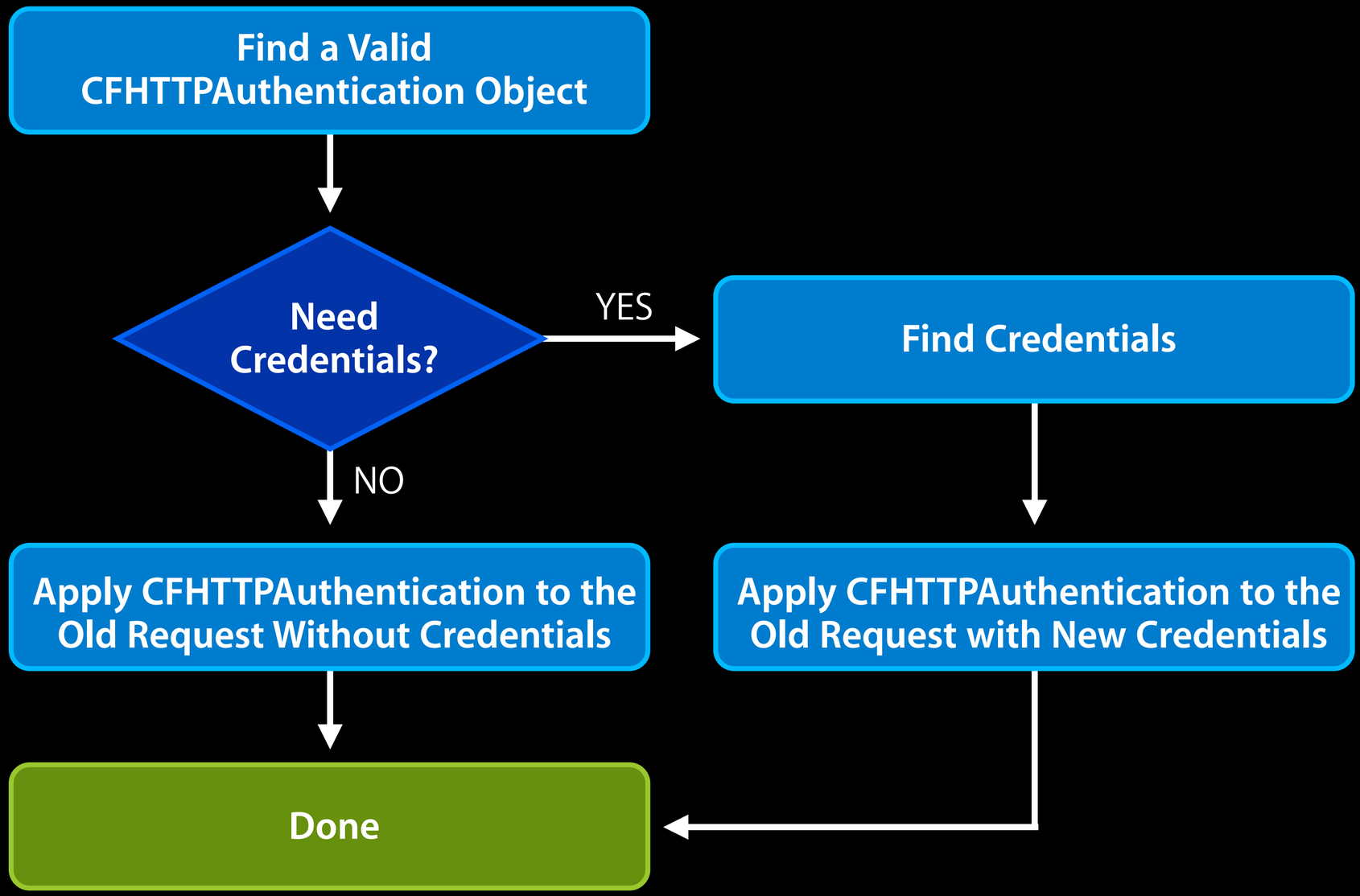
- Tracks the authentication dialog with a server
  - Enables multi-leg authentication types like NTLM
  - Keeps state to streamline future requests
- Does NOT hold the actual credentials
  - Getting and storing the credentials is your responsibility
- Apply to a request to get the new authentication-ready request
  - If credentials are needed, apply them at this time

Using CFHTTPAuthentication

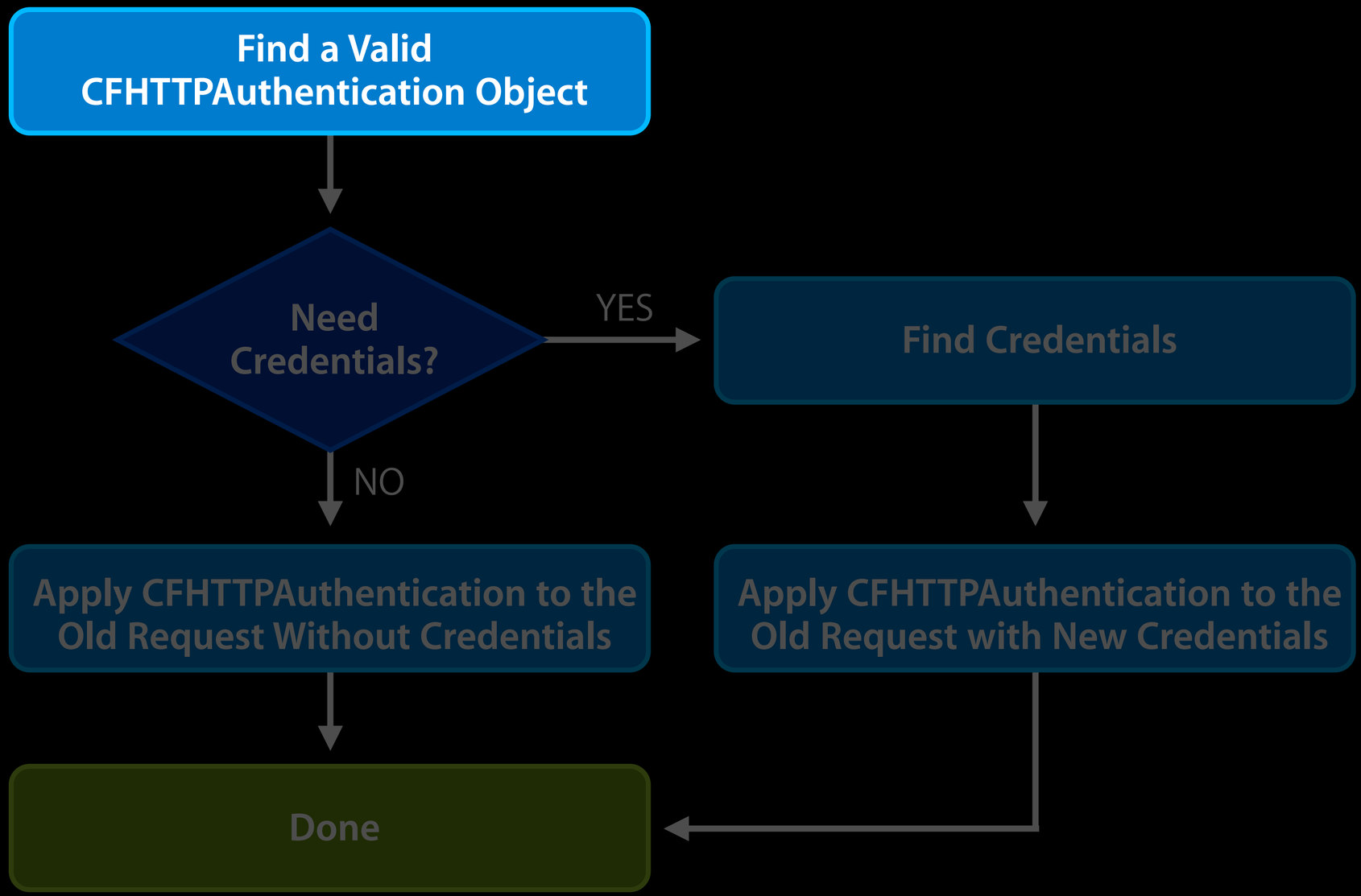
Three rules

- Use an authentication object until it becomes invalid
- Use only one set of credentials per authentication object
- When an authentication object becomes invalid, try to create a new one
  - If you get NULL, CFNetwork couldn't understand the server's challenge
  - If you get an invalid CFHTTPAuthentication, the server is forbidding future attempts
  - If you get a valid CFHTTPAuthentication, you may try again with new credentials

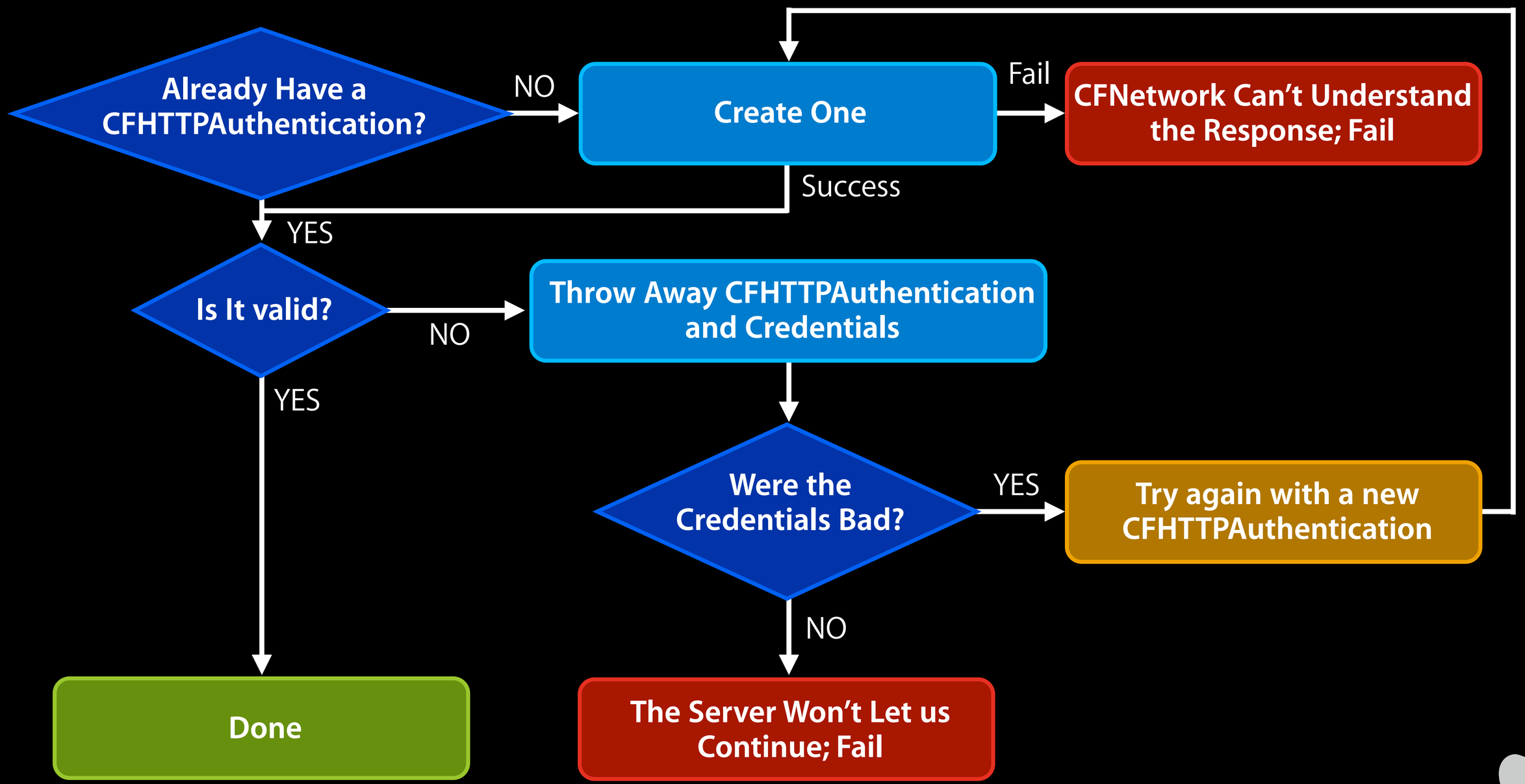
Modifying a Request with CFHTTPAuthentication



Modifying a Request with CFHTTPAuthentication



Finding a Valid CFHTTPAuthentication Object



Handling Authentication—CFNetworkLoader

Finding a valid CFHTTPAuthentication

```
- (void)retryAfterAuthorizationFailure {

  if (!authentication)
    createAuthentication;

  if (!authentication)
    fail; // Could not create authentication

  if (!valid(authentication)) {
    destroyAuthenticationAndCredentials;

    if (failureWasBadCredentials) {
      tryAgain;
    } else {
      fail; // The server won’t talk to us
    }
  } else {
    ... use authentication ...
  }
}
```

Handling Authentication—CFNetworkLoader

Finding a valid CFHTTPAuthentication

```
- (void)retryAfterAuthorizationFailure {

  if (!authentication)
    createAuthentication;

  if (!authentication)
    fail; // Could not create authentication

  if (!valid(authentication)) {
    destroyAuthenticationAndCredentials;

    if (failureWasBadCredentials) {
      tryAgain;
    } else {
      fail; // The server won’t talk to us
    }
  } else {
    ... use authentication ...
  }
}
```

Handling Authentication—CFNetworkLoader

Finding a valid CFHTTPAuthentication

```
- (void)retryAfterAuthorizationFailure {

  if (!authentication) {
    // Try to create one
    CFHTTPMessageRef responseHeader =
      CFReadStreamCopyProperty(readStream, kCFStreamPropertyHTTPResponseHeader);

  authentication = CFHTTPAuthenticationCreateFromResponse(NULL, responseHeader);

  CFRelease(responseHeader);
}

...
```

Handling Authentication—CFNetworkLoader

Finding a valid CFHTTPAuthentication

```
- (void)retryAfterAuthorizationFailure {

  if (!authentication)
    createAuthentication;

  if (!authentication)
    fail; // Could not create authentication

  if (!valid(authentication)) {
    destroyAuthenticationAndCredentials;

    if (failureWasBadCredentials) {
      tryAgain;
    } else {
      fail; // The server won’t talk to us
    }
  } else {
    ... use the authentication object ...
  }
}
```

Handling Authentication—CFNetworkLoader

Finding a valid CFHTTPAuthentication

```
- (void)retryAfterAuthorizationFailure {

  ...

  if (!authentication) {

    // Couldn’t create an authentication object
    [imageClient errorOccurredLoadingImage:NULL];

  } else if (!CFHTTPAuthenticationIsValid(authentication, &err)) {

    ...
```

Handling Authentication—CFNetworkLoader

Finding a valid CFHTTPAuthentication

```
- (void)retryAfterAuthorizationFailure {

  if (!authentication)
    createAuthentication;

  if (!authentication)
    fail; // Could not create authentication

  if (!valid(authentication)) {
    destroyAuthenticationAndCredentials;

    if (failureWasBadCredentials) {
      tryAgain;
    } else {
      fail; // The server won't talk to us
    }
  } else {
    ... use authentication ...
  }
}
```

Handling Authentication—CFNetworkLoader

Finding a valid CFHTTPAuthentication

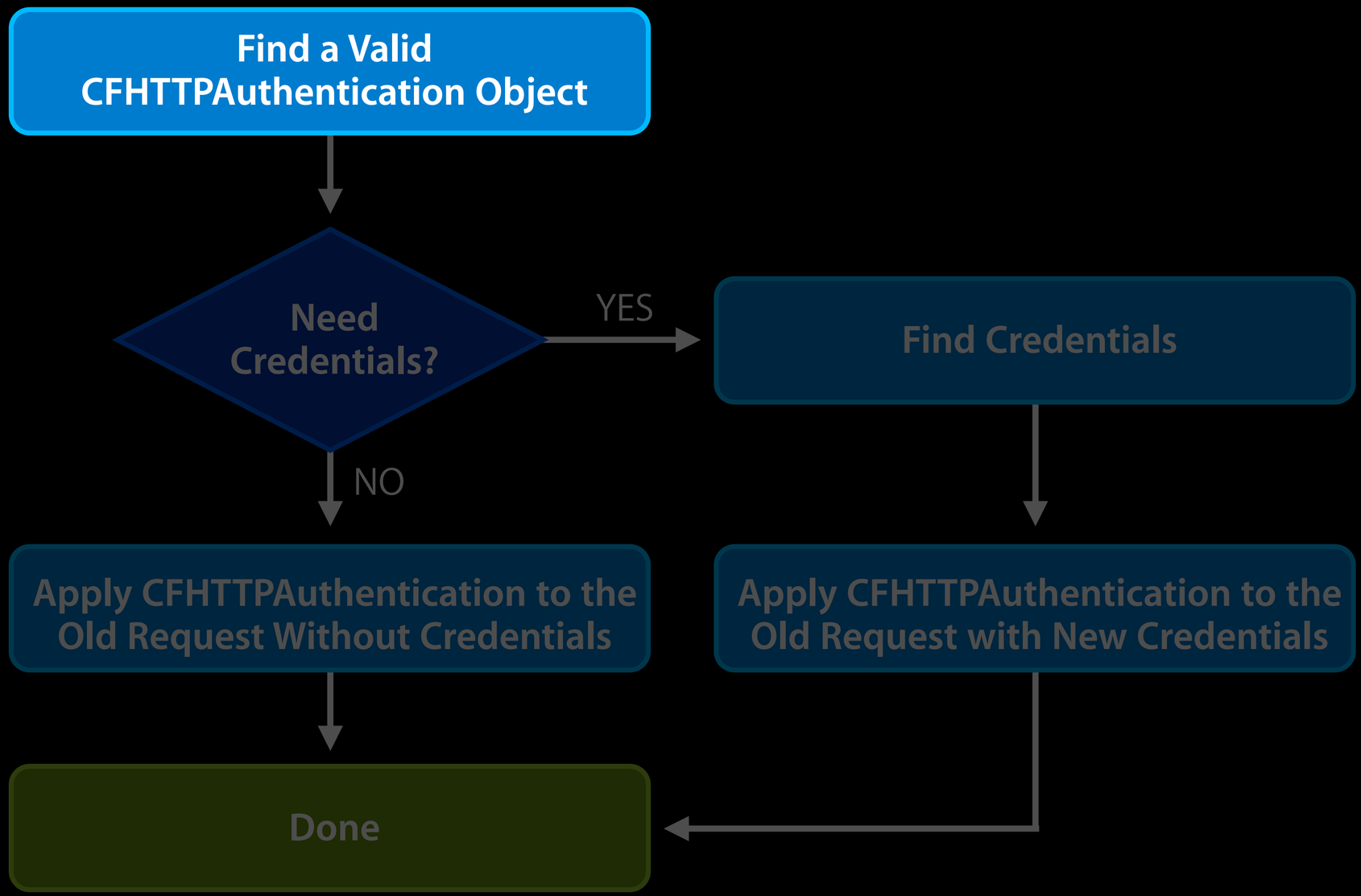
```
    ...

  } else if (!CFHTTPAuthenticationIsValid(authentication, &err)) {

    ... Destroy credentials and authentication ...

    // Check for bad credentials and treat these separately
    if (err.domain == kCFStreamErrorDomainHTTP &&
      (err.error == kCFStreamErrorHTTPAuthenticationBadUserName
      || err.error == kCFStreamErrorHTTPAuthenticationBadPassword)) {

      [self retryAfterAuthorizationFailure];
      return;

    } else {
      [imageClient errorOccurredLoadingImage:NULL];
    }

  } else {
```

Handling Authentication—CFNetworkLoader

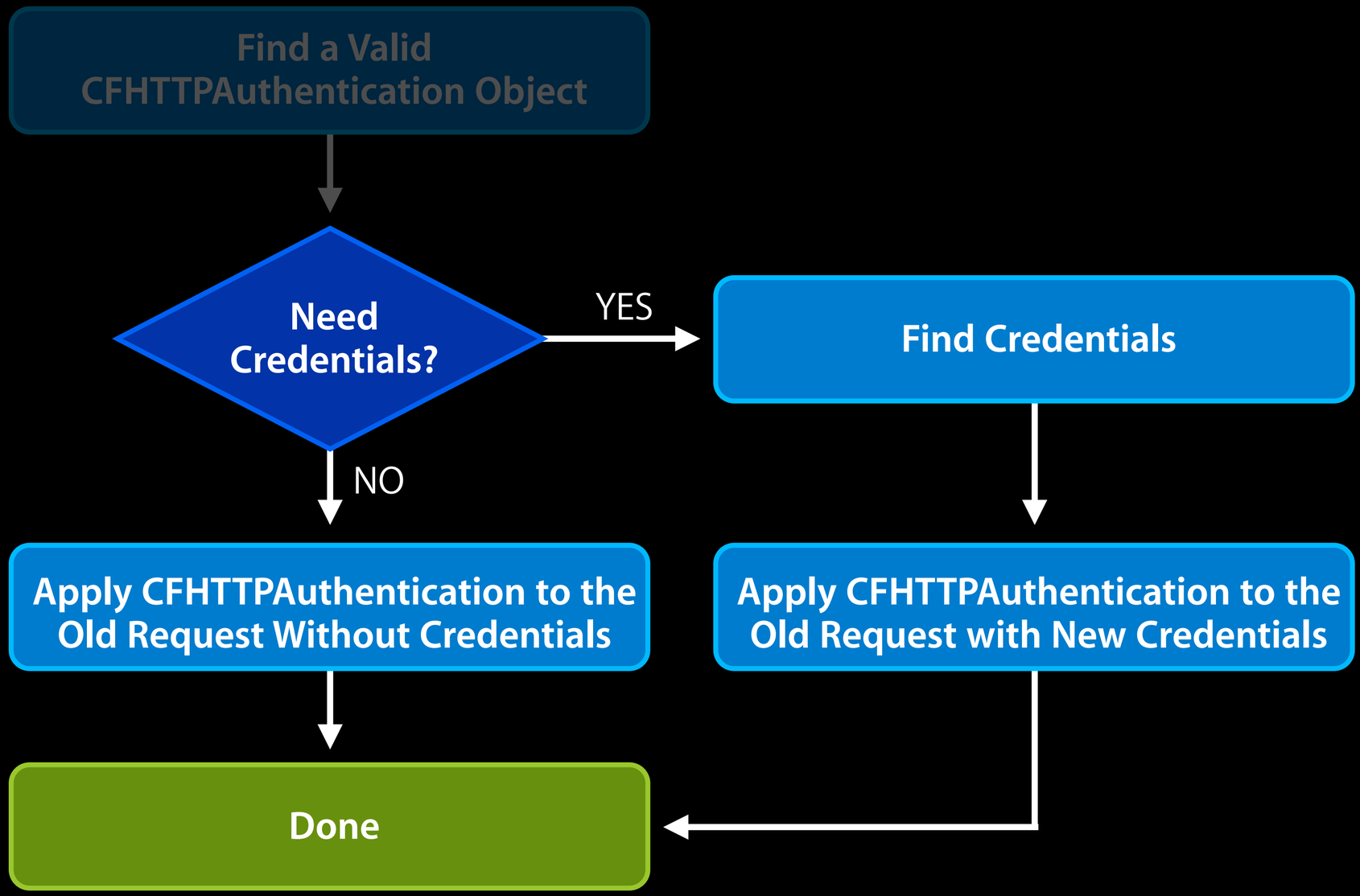
Finding a valid CFHTTPAuthentication

```
- (void)retryAfterAuthorizationFailure {

  if (!authentication)
    createAuthentication;

  if (!authentication)
    fail; // Could not create authentication

  if (!valid(authentication)) {
    destroyAuthenticationAndCredentials;

    if (failureWasBadCredentials) {
      tryAgain;
    } else {
      fail; // The server won't talk to us
    }
  } else {
    ... use authentication ...
  }
}
```

Modifying a Request with CFHTTPAuthentication



Modifying a Request with CFHTTPAuthentication



Handling Authentication—CFNetworkLoader

Using a CFHTTPAuthentication

```
- (void)retryAfterAuthorizationFailure {
  ... get a valid authentication object ...
    [self cancelLoad];

    if (credentials || !CFHTTPAuthenticationRequiresUserNameAndPassword(authentication)) {
      [self resumeWithCredentials];
    } else {
    CFStringRef realm = NULL;
    CFURLRef url = CFHTTPMessageCopyRequestURL(request);

    if (!CFHTTPAuthenticationRequiresAccountDomain(authentication)) {
      realm = CFHTTPAuthenticationCopyRealm(authentication);
    }

    [imageClient authorizationNeededForRealm: (NSString*)realm onHost: [(NSURL*)url host]];

    if (realm) CFRelease(realm);
    CFRelease(url);
  }
```

Handling Authentication—CFNetworkLoader

Using a CFHTTPAuthentication

```
- (void)retryAfterAuthorizationFailure {
  ... get a valid authentication object ...
    [self cancelLoad];

    if (credentials || !CFHTTPAuthenticationRequiresUserNameAndPassword(authentication)) {
      [self resumeWithCredentials];
    } else {
    CFStringRef realm = NULL;
    CFURLRef url = CFHTTPMessageCopyRequestURL(request);

    if (!CFHTTPAuthenticationRequiresAccountDomain(authentication)) {
      realm = CFHTTPAuthenticationCopyRealm(authentication);
    }

    [imageClient authorizationNeededForRealm: (NSString*)realm onHost: [(NSURL*)url host]];

    if (realm) CFRelease(realm);
    CFRelease(url);
  }
```

Handling Authentication—CFNetworkLoader

Using a CFHTTPAuthentication

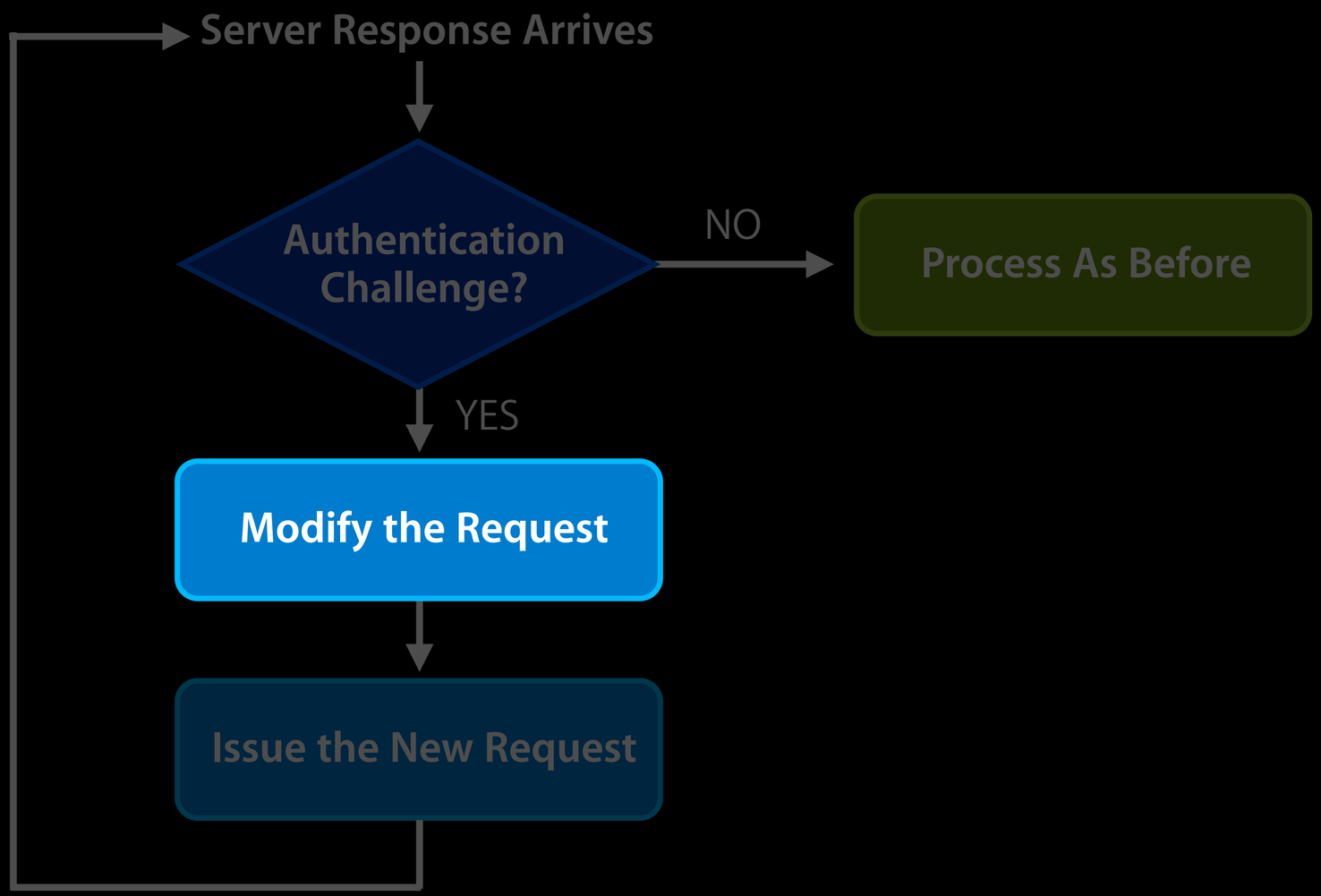
```
- (void)retryAfterAuthorizationFailure {
  ... get a valid authentication object ...
    [self cancelLoad];

    if (credentials || !CFHTTPAuthenticationRequiresUserNameAndPassword(authentication)) {
      [self resumeWithCredentials];
    } else {
    CFStringRef realm = NULL;
    CFURLRef url = CFHTTPMessageCopyRequestURL(request);

    if (!CFHTTPAuthenticationRequiresAccountDomain(authentication)) {
      realm = CFHTTPAuthenticationCopyRealm(authentication);
    }

    [imageClient authorizationNeededForRealm: (NSString*)realm onHost: [(NSURL*)url host]];

    if (realm) CFRelease(realm);
    CFRelease(url);
  }
```

Handling Authentication—CFNetworkLoader

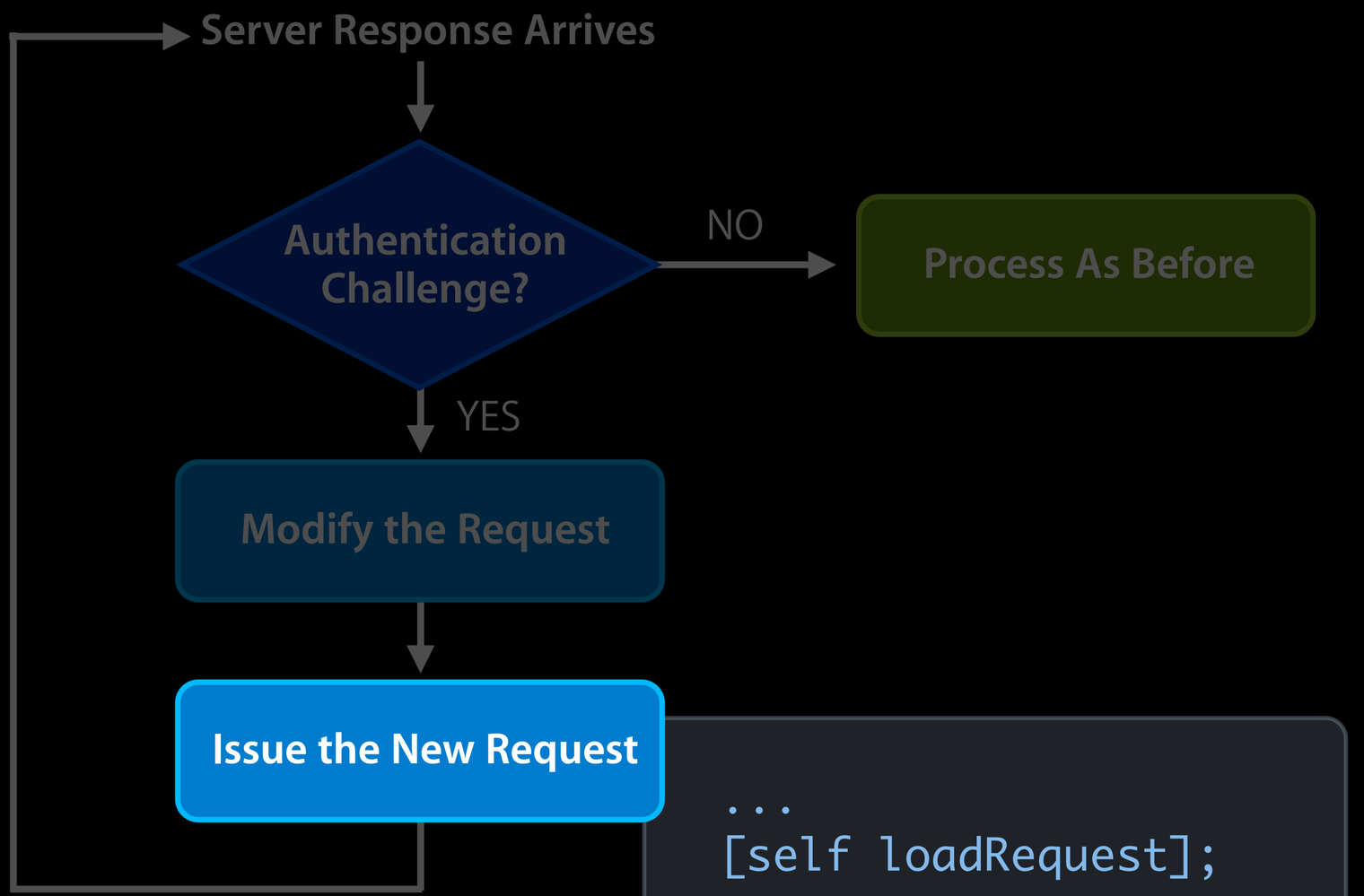
Using a CFHTTPAuthentication

```
- (void)resumeWithCredentials {
  credentials = CFDictionaryCreateMutable(NULL, 0,
                  &kCFTypeDictionaryKeyCallBacks, &kCFTypeDictionaryValueCallBacks);

  if (CFHTTPAuthenticationRequiresUserNameAndPassword(authentication)) {

    CFDictionarySetValue(credentials, kCFHTTPAuthenticationUsername, [imageClient username]);
    CFDictionarySetValue(credentials, kCFHTTPAuthenticationPassword, [imageClient password]);

    if (CFHTTPAuthenticationRequiresAccountDomain(authentication)) {
      CFDictionarySetValue(credentials, kCFHTTPAuthenticationAccountDomain,
          [imageClient accountDomain]);
    }
  }

  if (!CFHTTPMessageApplyCredentialDictionary(request, authentication, credentials, NULL)) {
    [imageClient errorOccurredLoadingImage:NULL];
  } else {
    [self loadRequest];
  }
}
```

Handling Authentication—CFNetworkLoader



Handling Authentication—CFNetworkLoader




```
...
[self loadRequest];
...
```

Demo

Authentication in Image Client

http://developer.apple.com/wwdc2005/

Step 5: Reusing Credentials

Storing credentials and recognizing when to reuse them

http://developer.apple.com/wwdc2005/

Reusing Credentials

- Reuse credentials to avoid re-prompting for username and password
- Watch out for security issues storing passwords!
- ImageClient stores credentials two places:
  - In memory
  - In the Keychain

Reusing Credentials

- Reuse credentials to avoid re-prompting for username and password
- Watch out for security issues storing passwords!
- ImageClient stores credentials two places:
  - In memory
  - In the Keychain

Storing Credentials—NSURLLoader

Nothing to do!

```
- (void)resumeWithCredentials {
    // Grab the username and password from imageClient and apply them to the challenge
    NSString *user = [imageClient username];
    NSString *pass = [imageClient password];

    // Guarantee values
    if (!user) user = @"";
    if (!pass) pass = @"";

    NSURLCredential *creds = [NSURLCredential credentialWithUser:user password:pass
        persistence:NSURLCredentialPersistenceForSession];

    [[challenge sender] useCredential:creds forAuthenticationChallenge:challenge];

    [challenge release];
    challenge = nil;
}
```

Reusing Credentials—CFNetworkLoader

- Create a store to hold all authentication objects

```
CFMutableArrayRef authArray;
// replaces CFHTTPAuthenticationRef authentication;
```

- Create a mapping from authentication objects to credentials

```
CFMutableDictionaryRef credentialsDict;
// replaces CFMutableDictionaryRef credentials;
```

- Maintain these everywhere we used to modify the current authentication object and the current credentials

```
CFDictionaryRemoveValue(credentialsDict, authentication);
// replaces CFRelease(credentials);
```

Reusing Credentials—CFNetworkLoader

Finding authentication objects and credentials

- Look for a matching authentication object before every load

```
- (CFHTTPAuthenticationRef)findAuthenticationForRequest {
    ...
    for (i = 0; i < c; i ++) {
        CFHTTPAuthenticationRef auth = CFArrayGetValueAtIndex(authArray, i);
        if (CFHTTPAuthenticationAppliesToRequest(auth, request)) {
            return auth;
        }
    }
    ...
}
```

- Before prompting the user, look for credentials in the credentials store

```
credentials = CFDictionaryGetValue(credentialsDict, authentication);
```

WARNING: Do NOT apply credentials before receiving an server challenge!

Step 6: Keeping Credentials

Storing credentials in the Keychain

http://developer.apple.com/wwdc2005/

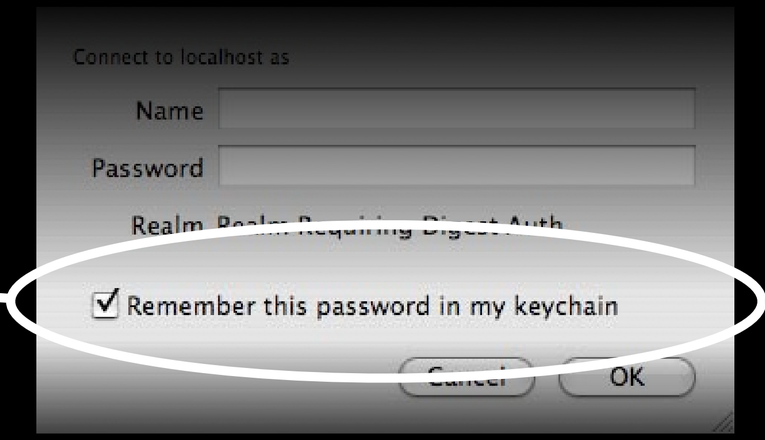
Keeping Credentials In the Keychain

- Keychain provides
  - Secure, persistent storage for credentials
  - Shared credentials across Safari, Mail, WebDAV…
- In-memory storage becomes a local cache
- Keychain becomes the persistent store

Keeping Credentials—NSURLLoader

- ImageClient

```
- (BOOL)saveCredentials;
```



- NSURLLoader

```
- (void)resumeWithCredentials {
  ...
  NSURLCredentialPersistence persistence = NSURLCredentialPersistenceForSession;
  if ([imageClient saveCredentials]) {
    persistence = NSURLCredentialPersistencePermanent;
  }

  NSURLCredential *creds =
    [NSURLCredential credentialWithUser:user password:pass persistence:persistence];
   [[challenge sender] useCredential:creds forAuthenticationChallenge:challenge];
   ...
}
```

Keeping Credentials—CFNetworkLoader

Two new methods

```
// Find the credentials dictionary for authentication
- (CFMutableDictionaryRef)
    findCredentialsForAuthentication:(CFHTTPAuthenticationRef)auth;

// Will find and save the credentials for the recent request.
- (void)saveCredentialsForRequest;
```

Keeping Credentials—CFNetworkLoader

Finding credentials in the Keychain

1. Check your local store first
2. Then call SecKeychainFindInternetPassword() to get the password from the Keychain
   - Retrieves a SecKeychainItem
   - Most of the arguments are taken from the URL
   - The realm and authentication method are taken from the CFHTTPAuthentication object
3. With a SecKeychainItem, call SecKeychainItemCopyContent to retrieve the username and password

Keeping Credentials—CFNetworkLoader

Finding credentials in the Keychain

1. Check your local store first
2. Then call SecKeychainFindInternetPassword() to get the password from the Keychain
   - Retrieves a SecKeychainItem
   - Most of the arguments are taken from the URL
   - The realm and authentication method are taken from the CFHTTPAuthentication object
3. With a SecKeychainItem, call SecKeychainItemCopyContent to retrieve the username and password

Keeping Credentials—CFNetworkLoader

Finding credentials in the Keychain

```
- (CFMutableDictionaryRef)findCredentialsForAuthentication:(CFHTTPAuthenticationRef)auth {

   // Check for cached credentials
   CFMutableDictionaryRef credentials = CFDictionaryGetValue(credentialsDict, auth);

   // Not cached, so go to Keychain.
   if (!ignoreKeychain && !credentials && CFHTTPAuthenticationRequiresUserNameAndPassword(auth)) {
     ...
   }
}
```

Keeping Credentials—CFNetworkLoader

Finding credentials in the Keychain

1. Check your local store first
2. Then call SecKeychainFindInternetPassword() to get the password from the Keychain
   - Retrieves a SecKeychainItem
   - Most of the arguments are taken from the URL
   - The realm is taken from the CFHTTPAuthentication object
3. With a SecKeychainItem, call SecKeychainItemCopyContent to retrieve the username and password

Keeping Credentials—CFNetworkLoader

Finding credentials in the Keychain

```
- (CFMutableDictionaryRef)findCredentialsForAuthentication:(CFHTTPAuthenticationRef)auth {
   ...
    // Not cached, so go to Keychain.
    if (!ignoreKeychain && !credentials && CFHTTPAuthenticationRequiresUserNameAndPassword(auth)) {

      ... Marshal a bunch of arguments from the URL ...

        didFind = SecKeychainFindInternetPassword(NULL,
                    strlen(host), host,                     /* serverName */
                    realm ? strlen(realm) : 0, realm,       /* securityDomain */
                    0, NULL,                                /* no accountName */
                    path ? strlen(path) : 0, path,          /* path */
                    port,                                   /* port */
                    protocolType,                           /* protocol */
                    authenticationType,                     /* authType */
                    0, NULL,                                /* no password */
                    &itemRef);

    if (didFind == noErr) {
      ...
    }
```

Keeping Credentials—CFNetworkLoader

Finding credentials in the Keychain

1. Check your local store first
2. Then call SecKeychainFindInternetPassword() to get the password from the Keychain
   - Retrieves a SecKeychainItem
   - Most of the arguments are taken from the URL
   - The realm is taken from the CFHTTPAuthentication object
3. With a SecKeychainItem, call SecKeychainItemCopyContent to retrieve the username and password

Keeping Credentials—CFNetworkLoader

Finding credentials in the Keychain

```
if (didFind == noErr) {
  ...

  // We want the account name attribute
  attr.tag = kSecAccountItemAttr;
  attr.length = 0;
  attr.data = NULL;
  attrList.count = 1;
  attrList.attr = &attr;

  if (SecKeychainItemCopyContent(itemRef, NULL, &attrList, &length, &outData) == noErr) {
    ...
  }
}
```

Keeping Credentials—CFNetworkLoader

Finding credentials in the Keychain

```
if (SecKeychainItemCopyContent(itemRef, NULL, &attrList, &length, &outData) == noErr) {

  // attr.data is the account (username) and outdata is the password
  CFStringRef username = CFStringCreateWithBytes(kCFAllocatorDefault,
                            attr.data, attr.length, kCFStringEncodingUTF8, false);
  CFStringRef password = CFStringCreateWithBytes(kCFAllocatorDefault,
                            outData, length, kCFStringEncodingUTF8, false);

  SecKeychainItemFreeContent(&attrList, outData);

  credentials = CFDictionaryCreateMutable(NULL, 0, &kCFTypeDictionaryKeyCallBacks,
                                          &kCFTypeDictionaryValueCallBacks);
  CFDictionarySetValue(credentials, kCFHTTPAuthenticationUsername, username);
  CFDictionarySetValue(credentials, kCFHTTPAuthenticationPassword, password);

  CFRelease(username);
  CFRelease(password);
}
```

Keeping Credentials—CFNetworkLoader

Storing credentials in the Keychain

1. Use SecKeychainFindInternetPassword() to find the correct item
   - Just like when finding credentials
   - EXCEPT supply the account name
2. If you find an item
   - Call SecKeychainItemModifyContent to update its content
3. If you don’t find an item
   - Call SecKeychainAddInternetPassword() to add the new password

Keeping Credentials—CFNetworkLoader

Storing credentials in the Keychain

```
- (void)saveCredentialsForRequest {

   ... Marshal a bunch of arguments from the URL ...

   didFind = SecKeychainFindInternetPassword(NULL,
                strlen(host), host,                     /* serverName */
                realm ? strlen(realm) : 0, realm,       /* securityDomain */
                strlen(username), username,             /* accountName */
                path ? strlen(path) : 0, path,          /* path */
                port,                                   /* port */
                protocolType,                           /* protocol */
                authenticationType,                     /* authType */
                0, NULL,                                /* no password */
                 &itemRef);

  if (didFind == noErr) {
    ... Modify this item ...
  } else {
    ... Create a new item ...
  }
```

Keeping Credentials—CFNetworkLoader

Storing credentials in the Keychain

```
didFind = SecKeychainFindInternetPassword( ... );
if (didFind == noErr) {

  SecKeychainAttribute attr;
  SecKeychainAttributeList attrList;

  // The attribute we want is the account name
  attr.tag = kSecAccountItemAttr;
  attr.length = strlen(username);
  attr.data = (void*)username;

  attrList.count = 1;
  attrList.attr = &attr;

  // Want to modify so that Keychain entry metadata is preserved.
  SecKeychainItemModifyContent(itemRef, &attrList, strlen(password), (void *)password);
  CFRelease(itemRef);

} else {
  ... Create a new item ...
}
```

Keeping Credentials—CFNetworkLoader

Storing credentials in the Keychain

```
didFind = SecKeychainFindInternetPassword( ... );
if (didFind == noErr) {
  ... Modify this item ...
} else {

    SecKeychainAddInternetPassword(NULL,
                                   strlen(host), host,                  /* serverName */
                                   realm ? strlen(realm) : 0, realm,    /* securityDomain */
                                   strlen(username), username,          /* accountName */
                                   path ? strlen(path) : 0, path,       /* path */
                                   port,                                /* port */
                                   protocolType,                        /* protocol */
                                   authenticationType,                  /* authenticationType */
                                   strlen(password), password,          /* password */
                                   &itemRef);

    CFRelease(itemRef);

}
```

Demo

Storing credentials in the Keychain

http://developer.apple.com/wwdc2005/

Step 7: Authenticating Firewalls

Extending the authentication model to proxies

http://developer.apple.com/wwdc2005/

Authenticating Firewalls

Proxies may require authentication too

The good news…

…the code is largely the same as for origin server authentication

The bad news…

…every request must go through both code paths

Two Code Paths

- Every request needs to be checked for proxy authentication and server authentication
- Need separate stores for proxies and origin servers
  - You'll be using the proxy authentication over and over and over…
  - You may need credentials for both a proxy and an origin server
- If the proxy settings change, discard the proxy store
- Save new proxy credentials once past the proxy
  - Don't wait for the origin server to authenticate!

Proxy Authentication

Differences from server authentication

- User prompt should be different
  - The user needs to know you want the proxy password
- Proxy challenges are 407 HTTP responses
  - Server challenges are 401
- Arguments to the Keychain calls come from the proxy host and port
  - For the server, the arguments came from the URL

Demo

ImageClient in all its glory

http://developer.apple.com/wwdc2005/

More Information

http://developer.apple.com/wwdc2005/

Documentation, Sample Code, and Other Resources

http://developer.apple.com/wwdc2005/

Related Sessions

Lab: Networking APIs	Friday 9:00AM
Implementing Networking Using the Cocoa Classes	Session 132 Thursday 10:30AM
Designing Network-Friendly Applications	Session 630 Thursday 3:30PM

Who to Contact

Xavier Legros
Worldwide Developer Relations
xavier@apple.com

Mark Malone
Worldwide Developer Relations
mmalone@apple.com

Q&A

Mark Malone
Worldwide Developer Relations
mmalone@apple.com

http://developer.apple.com/wwdc2005/

WWDC2005

Worldwide Developers Conference